Dive into Code Quality, Hashing & Complexity

PRESENTED BY MARYAM ABDELRASHID – AUGUST 2025

AGENDA

- What is Clean Code?
- Hash Function and Its Properties
- Time Complexity
  - For Stack
  - For Queue
- Solutions for Collision in Hashing
  - Separate Chaining
  - Open Addressing
  - Linear & Quadratic Probing
- Searching in Arrays
  - At First
  - At Middle
  - At Last

❑ What is a Clean Code?

- Clean code is code that is easy to read, easy to understand, and easy to maintain.

It's written in a way that makes it clear what the code is doing, even for someone who didn't write it.

Why is Clean Code important?

- Easier to debug and fix errors
- Helps teams to collaborate smoothly
- Reduces messy and confusing code
- Makes it easy to add new features
- Looks professional and clean

Clean Code Rules:

- Use clear and meaningful names
- Keep functions short and focused
- Avoid code duplication
- Format your code properly (indentation, spacing)

❑ Hash Function and its properties.

A hash function takes input data and returns a fixed-size value (called a hash code)
It helps to store, search, and access data efficiently.

◆ Where is it used?

- Hash tables (dictionaries or maps)
- Cryptography (for secure data)
- Data structures like sets and caches

◆ Properties of a Good Hash Function:

- Deterministic
  Same input → always gives the same output
- Fast to compute
  Should return the result quickly
- Uniform distribution
  Spreads values evenly across the hash table
- Collision resistance
  Hard for two inputs to get the same hash
- Irreversible (mainly for cryptography)
  You can't reverse the hash to find the original input

❑ Complexity

For stack (LIFO)

Last item in → first out

- Push, Pop, Peek: all O(1)
- Why? All actions happen at the top
- Space: O(n)

For queue (FIFO)

First item in → first out

- Enqueue, Dequeue, Peek: all O(1)
- Why? Actions happen at front & back
- Space: O(n)

❑ Searching in Arrays (First,Mid,Last)

1.First Element

```
arr = [5, 10, 15, 20]
first = arr[0]  # → 5
```

2.Mid Element

Odd number of elements

```
arr = [10, 20, 30]
middle = arr[len(arr) // 2]  # → 20
```

Even number of elements

```
arr = [5, 10, 15, 20]
mid = arr[len(arr) // 2]  # → 15
```

3. Last Element

```
arr = [5, 10, 15, 20]
last = arr[-1]  # → 20
```

1.Solutions for Collision in Hashing

- When two different keys produce the same index from the hash function, it's called a collision.

1. We need ways to handle that collision to avoid data loss or overwriting.

1. Separate Chaining:

- Each index holds a list.
- If multiple keys go to the same index, they are stored together in a list.

2. Linear Probing

- If a position is already taken, search for the next empty spot.
- Keeps checking forward one step at a time.

3. Quadratic Probing

- Similar to linear probing, but steps increase by square numbers.
- Checks: index + 1^2, index + 2^2, index + 3^2.

4. Double Hashing

- Uses a second hash function to calculate a new step size.
- Helps reduce clustering and collisions.

THANK YOU FOR LISTENING.